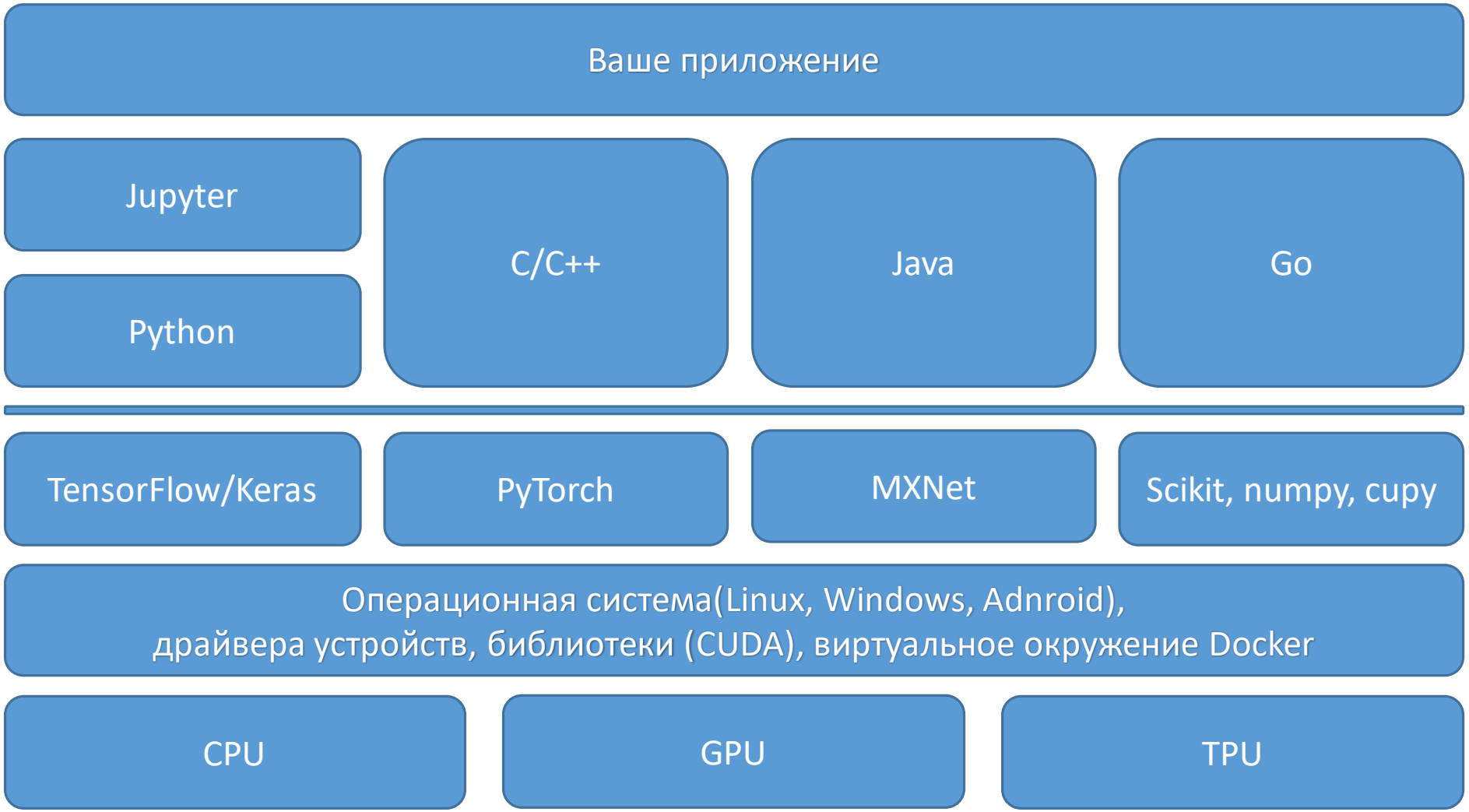
Семинар 1

Инструменты для машинного обучения

Стек технологий машинного обучения



Основы Python

Python – высокоуровневый мультипарадигменный интерпретируемый язык программирования общего назначения.
В данной презентации используется Python версии 3.7.2.

Установка:
Из conda (предпочтительный способ):
https://docs.conda.io/en/latest/miniconda.html
С официального сайта
https://www.python.org/downloads/

Управление пакетами (модулями):
1. Conda (***conda install numpy***)
2. Pip (***pip install numpy***)
3. Из исходников (***python setup.py install***)

Дзен Python

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если они не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один и, желательно, только один очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная штука! Будем делать их больше!

Основы Python: исполнение кода

Способы исполнения кода:

1. **Интерактивный режим интерпретатора: запуск интерпретатора python и ввод команд после символов >>>.**
- Быстрая проверка простых идей, до нескольких строк кода
2. **Сохраняем в файле .py (например, main.py) и выполняем команду python main.py [arguments].**
- Совместимость с IDE
- Удобство управления версиями
- Удобство автоматизации, возможность комбинировать разные программы в скриптах bash
3. **Интерактивный режим Jupyter Notebook с помощью веб-интерфейса.**
- Возможность произвольного запуска ячеек кода
- Встроенные возможности визуализации (графики, изображения)
4. **Интерактивный режим Jupyter Lab с помощью веб-интерфейса.**
- Все достоинства Jupyter Notebook
- Управление множеством ноутбуков jupyter в одной вкладке браузера
- Тёмная тема «из коробки».

Установка Jupyter Notebook: ***conda install jupyter*** (рекомендуется) или ***pip install jupyter***
Запуск Jupyter Notebook: ***jupyter notebook***
Установка Jupyter Lab: ***conda install jupyterlab*** (рекомендуется) или ***pip install jupyterlab***
Запуск Jupyter Lab: ***jupyter lab***

После выполнения этих команд в браузере откроется вкладка с веб-интерфейсом Jupyter.

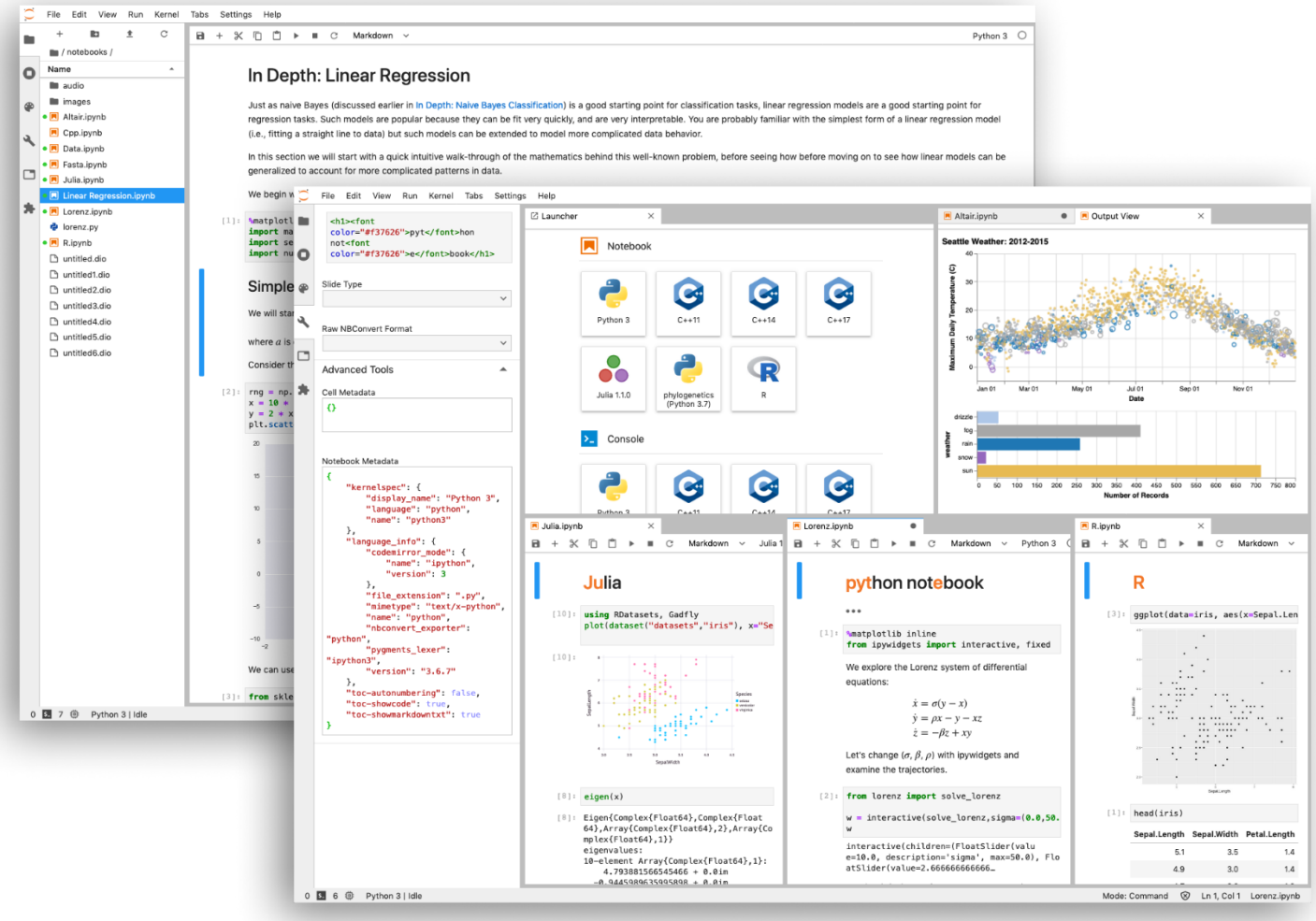
Основы Python: Notebook

Ноутбук состоит из ячеек (“cell”), которые бывают ячейками кода или разметки (например поясняющий текст или формулы).

Ядро IPython, выполняющееся в фоновом режиме хранит секущее состояние программы, которое меняется при исполнении ячеек кода.

Исполнение текущей (выделенной) ячейки производится нажатием кнопки “Run”. При зависании ядро IPython можно прервать (кнопка «interrupt») или перезагрузить (кнопка «Restart»). При прерывании состояние ядра сохраняется.

Системные команды можно выполнять в ячейке с помощью символа «!»: ***!pwd*** (печать текущей директории)



Основы Python: PEP 8

PEP 8 – это набор рекомендаций, описывающих стиль кода, а именно:

- Кодировка исходного кода (UTF-8)
- Правила импортирования модулей
- Максимальная длина строки (до 79 знаков для кода и до 72 для документации)
- Использование отступов – рекомендуется 4 пробела
- Использование пустых строк для разбивки кода на блоки
- Испроьзование комментариев
- Именование переменных, констант, классов, объектов, функций, модулей и т.д.

Ключевая идея состоит в том, что код читается гораздо больше раз, чем пишется, поэтому читаемость и прозрачность кода очень важна.

Ссылка на документ: https://www.python.org/dev/peps/pep-0008/

Основы Python: базовые функции

Особенности синтаксиса и базовые функции:

- Язык Python является регистрозависимым. A = 10 и a = 20 – это разные переменные.
- Печать в терминал или ноутбук осуществляется вызовом функции print: ***print("Hello, world!")***
- В отличии от C/C++, где блоки кода выделяются фигурными скобками, в Python блоки кода одного уровня всегда выделяются одинаковым числом отступов (пробелов или табов). Предпочтительный отступ – 4 пробела. Чередовать табы и пробелы в одной программе недопустимо в Python3.
- Однострочные комментарии начинаются с символа #. Многострочные комментарии обрамляются тройными кавычками """.
- Импорт модулей (пакетов) осуществляется командой import:
  - ***import numpy***
  - ***import numpy as np***
  - ***from numpy import * (не рекомендуется!!!)***
  - ***from numpy import max***
- Основные типы данных.
  - В Python3 всё является объектом какого-то класса
  - Числовые типы: int, float, complex
  - Логический: bool
  - Строковый: str
  - NoneType с единственным представителем None
  - Узнать тип объекта: ***type(x)*** или ***x.__class__***

```
>>> a = 100
>>> print(a, type(a))
100 <class 'int'>
>>> b = 1.234567
>>> print(b, type(b))
1.234567 <class 'float'>
>>> c = 1.234567e4
>>> print(c, type(c))
12345.67 <class 'float'>
>>> d = 1.6 + 10j
>>> print(d, type(d))
(1.6+10j) <class 'complex'>
>>> e = True
>>> print(e, type(e))
True <class 'bool'>
>>> f = a == 100
>>> print(f, type(f))
True <class 'bool'>
>>> g = "test"
>>> print(g, type(g))
test <class 'str'>
>>> h = None
>>> print(h, type(h))
None <class 'NoneType'>
```

Основы Python: структуры данных

Особенности синтаксиса и базовые функции:

- Основные структуры данных:
  - Кортеж (***tuple***) – неизменяемая последовательность данных любого типа
  - Список (***list***) – изменяемая последовательность данных любого типа
  - Словарь (***dict***) – ассоциативный массив, позволяющий получить доступ или добавить элемент по его ключу. Ключом может быть любой объект неизменяемого типа.
  - Множества (***set***) – неупорядоченный набор уникальных элементов.
- Индексация начинается с 0
- Присваивание переименых производится по ссылке. Переменные, ссылающиеся на один и тот же объект, имеют одинаковые идентификаторы. Уникальный идентификатор объекта можно получить так: ***id(x)***. Копирование по значению (т.е. создание нового объекта и копирование данных) производится методами ***copy(x)*** и ***deepcopy(x)***.

```
>>> # Создание кортежа
... tup1 = (1, 2, 3) # канонический способ
>>> tup2 = 1, 2, 3 # создание "на лету"
>>> some_list = [1, 2, 3] # список
>>> tup3 = tuple(some_list) # создание из списка (см. ниже)
>>> print(type(tup1), tup1, tup1 == tup2, tup2 == tup3)
<class 'tuple'> (1, 2, 3) True True
>>> # Распаковка кортежа в скалярные переменные
... a, b, c = tup1
>>> print(c, b, a)
3 2 1
>>> # Классическое применение кортежей:
... # поменять местами значения двух или более переменных
... b, a = a, b
>>> print(c, b, a)
3 1 2
>>> l = len(tup1) # Длина кортежа
>>> print(l)
3
```

```
>>> mylist = [[0]]*10
>>> print(mylist)
[[0], [0], [0], [0], [0], [0], [0], [0], [0], [0]]
>>> mylist[0][0] = 1
>>> print(mylist)
[[1], [1], [1], [1], [1], [1], [1], [1], [1], [1]]
>>> mylist[0] = 100
>>> print(mylist)
[100, [1], [1], [1], [1], [1], [1], [1], [1], [1]]
```

Основы Python: работа со списками

- Примеры работы со списками:
  - Срезы **(slice)** списков: ***s[start:end:step]***
  - Индекс start включается в срез, индекс end – не включается, step может быть отрицательным
  - Можно присваивать значения элементов по индексу или срезу
  - Удаление осуществляется вызовом ***del*** или присвоением пустого списка
  - Сортировка осуществляется методом ***sorted***, сложность O(n log n)
  - Добавить элемент к массиву можно методом ***append***
  - Склеить два массива можно оператором ***+*** или методом ***extend***

```
>>> s = [1, 2, 3, 4, 5]
>>> a = s[-1] # Последний элемент списка
>>> print(a)
5
>>> w = s[1:3]
>>> print(w)
[2, 3]
>>> # Можно опускать начальный индекс
... w = s[:2]
>>> print(w)
[1, 2]
>>> w = s[3:] # И конечный
>>> print(w)
[4, 5]
>>> w = s[::2] # И даже оба индекса
>>> print(w)
[1, 3, 5]
>>> w = s[::-1] # Инвертируем порядок
>>> print(w)
[5, 4, 3, 2, 1]
>>> # Заменяем элементы списка
... s[1:3] = [100, 200]
>>> print(s)
[1, 100, 200, 4, 5]
>>> s[2:4] = [] # Удаляем элементы списка
>>> print(s)
[1, 100, 5]
>>> del s[1:3] # Удаляем элементы списка
>>> print(s)
[1]
```

```
>>> s = [5, 3, 2, 4, 1]
>>> s1 = sorted(s)
>>> print(s1)
[1, 2, 3, 4, 5]
>>> s2 = sorted(s, reverse=True)
>>> print(s2)
[5, 4, 3, 2, 1]
>>> s2.append(0)
>>> print(s2)
[5, 4, 3, 2, 1, 0]
>>> s1.extend([6, 7, 8])
>>> print(s1)
[1, 2, 3, 4, 5, 6, 7, 8]
>>> s3 = s + ["a", "b", "c"]
>>> print(s3)
[5, 3, 2, 4, 1, 'a', 'b', 'c']
```

Основы Python: работа со словарями

- Примеры работы со словарями:
  - Словари можно создавать конструкцией ***{key0: value0, key1: value1, …}***.
  - Словари можно создавать передачей функции ***dict*** списка кортежей (key, value).
  - Доступ по ключу: ***d[key]***
  - Список ключей словаря: ***d.keys()***
  - Список значений словаря: ***d.values()***
  - Удаление элемента: ***del d[key]***
  - Проверка существования элемента с данным ключом: ***key in dictionary***

```
>>> # Создание словаря
... h1 = {'key1' : 1, 'key2' : 'val2', 11 : [32, 33]} # Обычный метод
>>> key_list = ['key1', 'key2', 11]
>>> value_list = [1, 'val2', [32, 33]]
>>> # Метод создания с помощью отдельных списков ключей и значений
... h2 = dict(zip(key_list, value_list))
>>> print(h1 == h2, h1)
True {'key1': 1, 'key2': 'val2', 11: [32, 33]}
>>> h2['key2'] = 'new_val' # Доступ по ключу
>>> print(h2)
{'key1': 1, 'key2': 'new_val', 11: [32, 33]}
>>>
>>> h2_keys = list(h2.keys()) # Список ключей словаря
>>> h2_values = list(h2.values()) # Список значений словаря
>>> print('Keys:', h2_keys, 'Values:', h2_values)
Keys: ['key1', 'key2', 11] Values: [1, 'new_val', [32, 33]]
>>> del h2['key2'] # Удаление элемента п
>>> print(h2)
{'key1': 1, 11: [32, 33]}
>>> print(len(h2)) # Длина словаря - количество пар (ключ, значение)
2
>>> key_in_h = 'key1' in h2 # Проверка на существование элемента с таким ключом
>>> print(key_in_h)
True
```

Основы Python: работа с множествами

- Примеры работы со словарями:
  - Множества создаются путём передачи списка элементов [e0, e1, …] функции set или перечислением элементов в фигурных скобках. При этом множество будет содержать только уникальные элементы
  - Объединение множеств осуществляется методом union или символом |
  - Пересечение множеств: ***a & b***
  - Элементы из множества a, но не b: ***a – b***
  - Исключающее ИЛИ: ***a ^ b***
  - Проверка существования элемента: ***element in set***

```
>>> s1 = set([1, 2, 3, 4, 5])
>>> print(type(s1))
<class 'set'>
>>> print(s1)
{1, 2, 3, 4, 5}
>>> s2 = {6, 7, 3, 4, 5, 3, 4, 5}
>>> print(s2)
{3, 4, 5, 6, 7}
>>> s3 = s1 | s2
>>> print(s3)
{1, 2, 3, 4, 5, 6, 7}
>>> s4 = s1 - s2
>>> print(s4)
{1, 2}
>>> s5 = s1 & s2
>>> print(s5)
{3, 4, 5}
>>> s6 = s1 ^ s2
>>> print(s6)
{1, 2, 6, 7}
>>> print(2 in s1)
True
```

Основы Python: функции

Особенности синтаксиса и базовые функции:

- Функции определяются с помощью ключевого слова ***def***. Возвращаемые значения определяются служебным словом ***return***. Можно вернуть несколько значений (кортеж). Если функция не заканчивается ***return***, то она возвращает значение ***None***.
- Параметры передаются в функцию по ссылке, но сами ссылки передаются по значению.
- Функция – это тоже объект
- Можно делать анонимные (лямбда) функции
- Аргументы функций бывают позиционными и именованными. Аргументы могут быть обязательными и необязательными.

```
>>> def foo(x, y, z=2):
...     return x + y + z
...
>>> foo(1, 2, 3)
6
>>> foo(1, 2)
5
>>> foo(x = 1, y = 2)
5
>>> foo(x = 1, z = 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: foo() missing 1 required positional argument: 'y'
```

```
>>> def foo(x):
...     x[0] = 1
...
>>> def bar(x):
...     x = [2]
...
>>> def baz(x):
...     x = [3]
...     return x
...
>>> foo_copy = foo
>>>
>>> x = [0, 0, 0]
>>> print(x)
[0, 0, 0]
>>> foo(x)
>>> print(x)
[1, 0, 0]
>>> bar(x)
>>> print(x)
[1, 0, 0]
>>> x = baz(x)
>>> print(x)
[3]
>>> foo_copy(x)
>>> print(x)
[1]
>>> my_lambda = lambda x : x**3
>>> x = 2
>>> print(my_lambda(x))
8
```

Основы Python: аргументы функций

Особенности синтаксиса и базовые функции:

- Функция может принимать произвольное количество позиционных аргументов. В этом случае перед именем параметра ставится * и этот параметр интерпретируется как кортеж.
- Функция может принимать произвольное количество именованных аргументов. В этом случае перед именем параметра ставится ** и этот параметр интерпретируется как словарь.

```
>>> def foo(*args):
...     print(type(args))
...     print(args)
...
>>> foo(1, 2, 3, "test")
<class 'tuple'>
(1, 2, 3, 'test')
>>> foo()
<class 'tuple'>
()
>>>
>>> def bar(**args):
...     print(type(args))
...     print(args)
...
>>> bar(a=1, b=2, c=3, d="test")
<class 'dict'>
{'a': 1, 'b': 2, 'c': 3, 'd': 'test'}
>>> bar()
<class 'dict'>
{}
```

Основы Python: управляющие конструкции

Особенности синтаксиса:

- Условный оператор задаётся шаблоном ***if-[elif]*-[else]?***
- Конструкции switch нет, вместо него используется ***if-elif.***
- Тернарный оператор: ***выражение1 if условие else выражение2***

```
>>> x = 23
>>> if x < 10:
...     print('x < 10')
... elif x < 30:
...     print('10 <= x < 30')
... else:
...     print('x >= 30')
...
10 <= x < 30
>>> print('x < 20') if x < 20 else print('x >= 20')
x >= 20
```

- Циклы с предусловием: ***while условие: <тело цикла> else: <действие>***

  Цикл выполняется, пока условие истинно. Действие внутри ветки else выполняется, когда условие цикла становится ложным.
- Переборный цикл: ***for i in итератор: <тело цикла> else: <действие>***

  Цикл выполняется для всех объектов в последовательности. Действие внутри ветки else выполняется, если цикл был завершён успешно.
- В обоих случаях ветка ***else*** не выполняется, если цикл был завершён с помощью ***break*** или произошло исключение.

```
>>> i = 3
>>> while i > 0:
...     print("Inside while loop: ", i)
...     i -= 1
... else:
...     print("Inside else branch: ", i)
...
Inside while loop:  3
Inside while loop:  2
Inside while loop:  1
Inside else branch:  0
>>> lis = [1, 2, 3]
>>> for i in lis:
...     print("Inside for loop: ", i)
... else:
...     print("Inside else branch: ", i)
...
Inside for loop:  1
Inside for loop:  2
Inside for loop:  3
Inside else branch:  3
```

Основы Python: Некоторые возможности циклов

Особенности синтаксиса:

- Ходом выполнения цикла можно управлять с помощью служебных слов ***break*** и ***continue***

  break: прекратить выполнение цикла

  continue: перейти к следующей итерации цикла
- В цикле можно производить распаковку кортежей
- Распаковку словарей можно осуществлять методом ***словарь.items()*** или ***in***.
- List comprehensions – однострочное формирование списка с проверкой условий.

```
>>> lis1 = [1, 2, 3, 4, 5]
>>> lis2 = [x*x for x in lis1 if x > 2]
>>> print(lis2)
[9, 16, 25]
```

```
>>> i = 10
>>> while i > 0:
...     if i == 7: # Не печатаем 7
...         i -= 1
...         continue
...     print('while', i)
...     if i == 5: # Выходим при достижении 5
...         break
...     i -= 1
...
while 10
while 9
while 8
while 6
while 5
>>> lis = [(1, 2), (2, 3), (6, 7)]
>>> for i, j in lis: # Распаковка кортежа
...     print('({},{})'.format(i, j))
...
(1,2)
(2,3)
(6,7)
>>> h = {'key1' : 'val1', 111 : 222, 'key3' : 0}
>>> for key, val in h.items(): # Пример использования items()
...     print('h[{}] = {}'.format(key, val))
...
h[key1] = val1
h[111] = 222
h[key3] = 0
>>> # Но можно и так
... for key in h: # В словаре итератор работает по ключам
...     print('h[{}] = {}'.format(key, h[key]))
...
h[key1] = val1
h[111] = 222
h[key3] = 0
```

Основы Python: Функции enumerate, zip, range, hash

Особенности синтаксиса:

- Автоматическую индексацию элементов списка можно производить с помощью функции ***enumerate***
- Одновременную итерацию по нескольким итерируемым объектам можно производить с помощью функции ***zip.*** Эта функция возвращает итератор кортежей, где i-тый кортеж содержит i-тые элементы итреируемых объектов.
- Итерацию по последовательности 0..N-1 можно осуществить вызовом ***range***.
- Функция ***hash*** возвращает уникальный идентификатор переданного её объекта, неизменяемого типа.

```
>>> names = ["Alice", "Bob", "Carol", "Dave", "Eve"]
>>> # Словарь имён, ключи которого - индексы массива имён
... name_by_index = dict(enumerate(names))
uids = [i for i in range(1000, 1010)]
>>> print(name_by_index)
{0: 'Alice', 1: 'Bob', 2: 'Carol', 3: 'Dave', 4: 'Eve'}
>>>
>>> uids = [i for i in range(1000, 1010)]
>>> print(uids)
[1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009]
>>> # Словарь имён, ключи которого - созданные нами идентификаторы
... # Обратите внимание, что zip производит последовательность длины, меньшей из двух последовательностей
... name_by_uid = dict(zip(uids, names))
>>> print(name_by_uid)
{1000: 'Alice', 1001: 'Bob', 1002: 'Carol', 1003: 'Dave', 1004: 'Eve'}
>>>
>>> hashes = [hash(name) for name in names]
>>> # Словарь имён, ключи которого - созданные уникальные хэши объектов
... name_by_hash = dict(zip(hashes, names))
>>> print(name_by_hash)
{4815068403253482091: 'Alice', 8191186711451458332: 'Bob', -4129885004479586670: 'Carol', 8582906478397666224: 'Dave', -7239745768166601447: 'Eve'}
```

Основы Python: Строки

Особенности синтаксиса:

- Строки можно понимать как массив символов.
- Строки заключаются в одинарные, двойные или тройные кавычки
- Внутри строк, заключённых в одинарные кавычки могут быть неэкранированные символы двойных кавычек и наоборот.
- Тройные кавычки используются для многострочных текстов.
- Форматирование строк:
  - Классический метод (оператор %): ***s = "%s %.2f %d" % ("abra", 1.1111, 34)***
  - Метод format: ***s = "{0} {1:.2f} {2:d}".format("abra", 1.1111, 34)***
  - F-strings: ***s = f"abra {1.1111:.2f} {34:d}"***
- В Python3 кодировка по-умолчанию – UTF-8.
- Изменять символы в строке нельзя, но можно конвертировать её в список, изменить его, а затем конвертировать обратно в строку

```
>>> s = "abc'"
>>> print(s)
abc'
>>> s = 'abc"'
>>> print(s)
abc"
>>> s = """abc
... xyz"""
>>> print(s)
abc
xyz
>>> s = "проверка"
>>> print(s)
проверка
```

```
>>> s = "%s %.2f %d" % ("abra", 1.1111, 34)
>>> print(s)
abra 1.11 34
>>> s = "{0} {1:.2f} {2:d}".format("abra", 1.1111, 34)
>>> print(s)
abra 1.11 34
>>> s = f"abra {1.1111:.2f} {34:d}"
>>> print(s)
abra 1.11 34
```

Основы Python: Строки

- С помощью функции ***join*** можно объединить последовательность в строку с заданным разделителем: ***разделитель.join(последовательность_строк)***
- С посощью функции ***split*** можно разделить строку в список по заданному разделителю: ***строка.split(разделитель)***
- Поиск подстроки: ***строка.find(подстрока)***
- Замена одной подстроки другой ***строка.replace(подстрока, замена)***
- Проверка нахождения подстроки в строке: ***подстрока in строка***
- Модуль регулярных выражений ***re*** – наиболее гибкий и мощный инструмент для работы со строками (но его использование не всегда оправдано)

```
>>> st_list1 = ['Hello', 'to', 'all']
>>> st1 = '_'.join(st_list1)
>>> print(st1)
Hello_to_all
>>>
>>> st2 = 'Hello,to,all'
>>> st_list2 = st2.split(',')
>>> print(st_list2)
['Hello', 'to', 'all']
>>>
>>> st = 'Hello to all'
>>> print('Index of "lo":', st.find('lo')) # индекс подстроки
Index of "lo": 3
>>> print('Index of "hi":', st.find('hi')) # или -1, если такой нет
Index of "hi": -1
>>>
>>> print('"to" is in string:', 'to' in st)
"to" is in string: True
>>> print('Replace "all" -> "you":', st.replace('all', 'you'))
Replace "all" -> "you": Hello to you
```

Основы Python: Работа с файлами

Особенности синтаксиса:

- В основе работы с файлами лежит встроенная функция open. Этой функции передаётся имя открываемого файла, режим открытия: ‘r’ – чтение, ‘w’ – запись, ‘a’ – дозапись, ‘b’ – бинарный режим (по-умолчанию - текстовый), ‘+’ – чтение и запись. Пример: ‘rb’ – чтение в бинарном режиме.
- Два способа работы с функцией open:

  # Способ 1

  f = open(filename, mode)

  <работаем с объектом f>

  f.close() # Важно не забыть закрыть файл

  # Способ 2 (рекомендуемый)

  with open(filename, mode) as f:

  <работаем с объектом f, закрытие осущетвится автоматически>
- Основные методы для работы с файлами:
  - ***read(size)*** – прочитать size (символов или байт)
  - ***write(data)*** – записать данные data в файл
  - ***tell()*** – текущая позиция в файле
  - ***seek(pos)*** – перейти на позицию pos
  - ***readline()*** – прочитать следующую строку
  - ***readlines() или list(f)*** – итерируемый список строк файла
- Работа с CSV таблицами осуществляется с помощью модуля ***csv***
- Работа с файлами JSON осуществляется с помощью модуля ***json***

```
>>> filename = "test.csv"
>>>
with open(filename, "w") as f:
>>> content = """0,Alice,Moscow
... 1,Bob,London
with open(filename, "r") as f:
... 2,Carol,Beijing"""
import csv
>>>
>>> with open(filename, "w") as f:
...     f.write(content)
...
43
    for idx, name, city in reader:
        print(f"{idx}: {name} from {city}")
>>> with open(filename, "r") as f:
...     for line in f.readlines():
...         print(f"Line: '{line}'".replace("\n", ""))
...
Line: '0,Alice,Moscow'
Line: '1,Bob,London'
Line: '2,Carol,Beijing'
>>> import csv
>>> with open(filename, "r") as f:
...     reader = csv.reader(f, delimiter=',')
...     for idx, name, city in reader:
...         print(f"{idx}: {name} from {city}")
...
0: Alice from Moscow
1: Bob from London
2: Carol from Beijing
```

Основы Python: Модуль sys

Модуль ***sys*** предоставляет доступ к функциям и переменным интерпретатора.

- ***sys.argv*** – список аргументов скрипта. ***sys.argv[0]*** содержит имя скрипта.
- ***sys.exit(arg)*** – завершение программы. Если ***arg*** – целое значение в интервале 0..127, то это значение будет кодом возврата процесса. Обычно значение 0 обозначает штатное завершение, иначе – код ошибки. Если ***arg*** – объект, отличный от целого значения, то он будет напечатан в консоль и возвращён код 1.
- ***sys.path*** – **очень важная переменная!** Содержит список путей поиска модулей. Если вы не смогли загрузить какой-то модуль, проверьте, есть ли путь к нему в этом списке, и, если нет, добавьте с помощью функции append().
- ***sys.platform*** – идентификатор платформы
- ***sys.version*** – версия интерпретатора и информация о сборке

```
>>> import sys
>>> print(sys.argv)
['-', 'arg1', 'arg2', 'arg3']
>>> print(sys.path)
['', '/opt/conda/lib/python37.zip', '/opt/conda/lib/python3.7', '/opt/conda/lib/python3.7/lib-dynload', '/opt/conda/lib/python3.7/site-packages']
>>> print(sys.platform)
linux
>>> print(sys.version)
3.7.5 (default, Oct 25 2019, 15:51:11)
[GCC 7.3.0]
>>> sys.exit(10)
```

Основы Python: Модуль os

Модуль ***os*** предоставляет интерфейс к различным функциям операционной системы. Модуль ***os.path*** предоставляет функции работы с именами файлов:

os.environ – контейнер (ассоциативный массив типа словаря), содержащий текущие переменные окружения, доступ к которым на чтение и запись можно получить по их ключу.

os.getcwd() – получить текущий рабочий каталог

os.chdir(path) – сменить текущий рабочий каталог

os.mkdir(path) – создать каталог

os.makedirs(path) – рекурсивное создание вложенных каталогов

os.system(cmd) – выполнить команду cmd

os.listdir(path) – список объектов в папке path

os.walk(path) – рекурсивный обход каталога

os.path.dirname(path) – имя директории объекта path

os.path.exists(path) – проверка существования объекта path

os.path.join(path, *paths) – «Правильное» объединение путей с использованием системных разделителей

os.path.abspath(path) – абсолютный путь к объекту path

```
>>> import os
>>> os.makedirs("/tmp/test/1/2/3")
>>> os.makedirs("/tmp/test/1/4")
>>> os.system("touch /tmp/test/5.txt")
0
>>> os.system("touch /tmp/test/6.txt")
0
>>> os.chdir("/tmp/test/")
>>>
>>> for root, dirs, files in os.walk("."):
...     print('Root:', os.path.abspath(root)) # Текущий каталог обхода
...     print('Files:', files) # Содержащиеся непосредственно в нем файлы
...     print('Dirs:', dirs) # Содержащиеся в нем директории первого уровня
...
Root: /tmp/test
Files: ['6.txt', '5.txt']
Dirs: ['1']
Root: /tmp/test/1
Files: []
Dirs: ['4', '2']
Root: /tmp/test/1/4
Files: []
Dirs: []
Root: /tmp/test/1/2
Files: []
Dirs: ['3']
Root: /tmp/test/1/2/3
Files: []
Dirs: []
```

Основы Python: обработка исключений

Особенности синтаксиса:

- Существует два классических способа обработки ошибок в программах:
  - В возвращаемом значении функции (в стиле Си)
  - Путём генерации исключений (C++, Python)
- Исключения (Exceptions) – это один из типов данных Python.

```
try:
    # Код, который может вызвать исключение
except КлассИсключения1:
    # Обработка исключения1
except КлассИсключения2:
    # Обработка исключения2
else:
    # Код, выполняемый если исключения не произошло
finally:
    # Код, выполняемый в любом случае
```

```
>>> def foo(a):
...     try:
...         k = 1 / a
...         m = 1000.0**a
...     except ZeroDivisionError:
...         print("Деление на ноль")
...     except OverflowError:
...         print("Ошибка переполнения")
...     else:
...         print(k, m)
...     finally:
...         print("Код, выполняемый в любом случае")
...
>>> foo(0)
Деление на ноль
Код, выполняемый в любом случае
>>> foo(10000000)
Ошибка переполнения
Код, выполняемый в любом случае
>>> foo(-0.1)
-10.0 0.5011872336272722
Код, выполняемый в любом случае
```

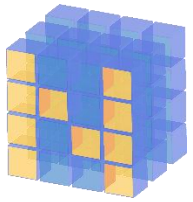
- Читайте тексты исключений! Если исключение не обрабатывается явно, то программа завершается, печатается бэктрейс исключения, в котором написано, в какой строке какого файла произошло исключения, класс этого исключения, как программа попала в эту строку (порядок вызова функций) и т.п. Зачастую этого достаточно для того, чтобы локализовать и исправить ошибку или, как минимум, «загуглить» её.

Основы Python: Объекты и классы

Особенности синтаксиса:

- Классы объявляются с помощью ключевого слова class
- Конструктор класса объявляется в методе __init__
- Деструктор класса объявляется в методе __del__
- Первым аргументом каждого нестатического метода должен быть объект self – т.е. сам объект
- Все аттрибуты и методы класса по-умолчанию являются публичными
- Аттрибуты и методы, начинающиеся с «_» по договорённости считаются приватными и относятся к реализации класса
- Все методы класса являются виртуальными
- Наследование: class SubClass(SuperClass)
- Функция super позволяет получить объект, приведённый к родительскому классу
- Метод __str__ определяет преобразование объекта в «читаемое» строковое представление, получаемое функцией str()
- Меотд __repr__ определяет представление объекта в «отладочное» представление, получаемое функцией repr()

```
>>> class SuperClass(object):
...     def __init__(self, x):
...         print("SuperClass constructor")
...         self.x = x
...     def __del__(self):
...         print("SuperClass destructor")
...     def __str__(self):
...         return(f"SuperClass[x={self.x}]")
...
>>> class SubClass(SuperClass):
...     def __init__(self, x, y):
...         super(SubClass, self).__init__(x)
...         print("SubClass constructor")
...         self.y = y
...     def __del__(self):
...         print("SubClass destructor")
...         super(SubClass, self).__del__()
...     def __str__(self):
...         return(f"SubClass[x={self.x}; y={self.y}]")
...     def __repr__(self):
...         return("This is an object of class SubClass")
...
>>> def foo():
...     c1 = SuperClass(100)
...     c2 = SubClass(200, 300)
...     print("c1 to string: " + str(c1))
...     print("c2 to string: " + str(c2))
...     print(repr(c1))
...     print(repr(c2))
...
>>> foo()
SuperClass constructor
SuperClass constructor
SubClass constructor
c1 to string: SuperClass[x=100]
c2 to string: SubClass[x=200; y=300]
<__main__.SuperClass object at 0x7f2cd898f390>
This is an object of class SubClass
SuperClass destructor
SubClass destructor
SuperClass destructor
```



Библиотека Numpy

Numpy – библиотека векторных, матричных и тензорных операций. Это один из главных инструментов в научных расчётах и машинном обучении.

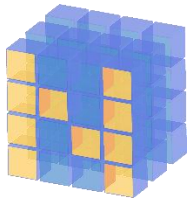
Особенности синтаксиса:

- Импорт модуля:
  import numpy as np
- Создание одномерного массива (вектора):
  a = np.array([1, 2, 3])
- Создание двумерного массива 2x3:
  b = np.array([[1.5, 2, 3], [4, 5, 6]])
- Создание трёхмерного массива 2x2x3:
  c = np.array([[[1.5, 2, 3], [4, 5, 6]],
  [[3, 2, 1], [4, 5, 6]]], dtype=np.float32)
- Массив нулей с заданной размерностью:
  x = np.zeros(shape=(3, 4))
- Трёхмерный массив из единиц:
  y = np.ones((2, 3, 4), dtype=np.int64)
- Напечатать размер массива:
  print(f"Shape of y is {y.shape}")

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
print(a)
b = np.array([[1.5, 2, 3], [4, 5, 6]])
>>> print(a)
c = np.array([[[1.5, 2, 3], [4, 5, 6]],
              [[3, 2, 1], [4, 5, 6]]], dtype=np.float32)
print(c)
x = np.zeros(shape=(3, 4))
print(x)
y = np.ones((2, 3, 4), dtype=np.int64)
print(y)
[1 2 3]
>>> b = np.array([[1.5, 2, 3], [4, 5, 6]])
>>> print(b)
[[1.5 2.  3. ]
 [4.  5.  6. ]]
>>> c = np.array([[[1.5, 2, 3], [4, 5, 6]],
...               [[3, 2, 1], [4, 5, 6]]], dtype=np.float32)
>>> print(c)
[[[1.5 2.  3. ]
  [4.  5.  6. ]]

 [[3.  2.  1. ]
  [4.  5.  6. ]]]
>>> x = np.zeros(shape=(3, 4))
>>> print(x)
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
>>> y = np.ones((2, 3, 4), dtype=np.int64)
>>> print(y)
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
>>> print(f"Shape of y is {y.shape}")
Shape of y is (2, 3, 4)
```



Библиотека Numpy

Особенности синтаксиса:

- Поэлементные операции выполняются операторами ***+, -, *, /, ****

x = np.array([[1, 2, 3], [4, 5, 6]])
print(x*3)

- Транспонирование осуществляется функцией ***np.transpose*** или свойством .T массива.

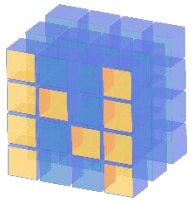
xT = x.T
xT_1 = np.transpose(x)

- Матричное умножение выполняется методом matmul или оператором ***@***
- Характеристики размера массива:
  - ***ndim***: количество размерностей (осей)
  - ***shape***: количество элементов по каждой из осей
  - ***len()***: количество элементов в первой оси
  - ***size***: полное количество элементов («объём»)
- Обратную матрицу можно найти функцией ***np.linalg.inv()***
- Определитель матрицы можно найти функцией ***np.linalg.det()***
- Систему линейных уравнений можно решить функцией ***np.linalg.solve()***

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(x)
[[1 2 3]
 [4 5 6]]
>>> print(x*3)
[[ 3  6  9]
 [12 15 18]]
>>> xT = np.transpose(x)
>>> print(xT)
[[1 4]
 [2 5]
 [3 6]]
>>> y = np.matmul(x, x.T)
>>> print(y)
[[14 32]
 [32 77]]
```

```
>>> c = np.array([[[1, 2, 3], [4, 5, 6]],
...               [[3, 2, 1], [4, 5, 6]]])
>>> print("Shape =", c.shape, "Length =", len(c))
Shape = (2, 2, 3) Length = 2
>>> print("NDim =", c.ndim, "Size =", c.size)
NDim = 3 Size = 12
```

```
>>> matr = np.array([[1, 2], [3, 4]])
>>> svob = np.array([1, 1])
>>> print('matr =', matr)
matr = [[1 2]
 [3 4]]
>>> print('matr^-1 =', np.linalg.inv(matr))
matr^-1 = [[-2.   1. ]
 [ 1.5 -0.5]]
>>> print('det(matr) =', np.linalg.det(matr))
det(matr) = -2.0000000000000004
>>> x = np.linalg.solve(matr, svob)
>>> print('x =', x)
x = [-1.  1.]
```



Библиотека Numpy: броадкастинг

Особенности синтаксиса:

- Под «броадкастингом» понимается механизм работы с массивами разных размерностей при выполнении поэлементных арифметических операций.
- Броадкастинг даёт средства векторизации операций над массивами (т.е. без явных циклов по осям)
- При работе с двумя массивами Numpy поэлементно сравнивает их формы (shape) от последнего элемента к первому. Две размерности совместимы, если:
  - Одни равны
  - Одна из них равна 1
- Если эти условия не выполнены, то будет сгенерировано исключение “ValueError: operands could not be broadcast together”
- При этом количество измерений не обязано быть одинаковым
- Правила индексации и срезов в целом аналогичны правилам для списков. Символ «:» означает «все элементы этой размерности». Символ «...» означает «Заменить все остальные координаты на :»
- Оператор np.newaxis позволяет добавить «виртуальную» ось единичного размера

```
>>> x = np.ones((256, 256, 3))
>>> y = np.array([1.0, 2.0, 3.0])
>>> z = x + y
>>> a = np.ones((256, 3))
>>> b = x + a
>>> c = np.ones((256, 1, 3))
>>> d = x + c
```

```
>>> a = np.array([0.0, 10.0, 20.0, 30.0])
>>> print(a.shape)
(4,)
>>> b = np.array([1.0, 2.0, 3.0])
>>> print(b.shape)
(3,)
>>> c = a[:, np.newaxis] + b
>>> print(c.shape)
(4, 3)
>>> print(c)
[[ 1.  2.  3.]
 [11. 12. 13.]
 [21. 22. 23.]
 [31. 32. 33.]]
```

Библиотека Matplotlib

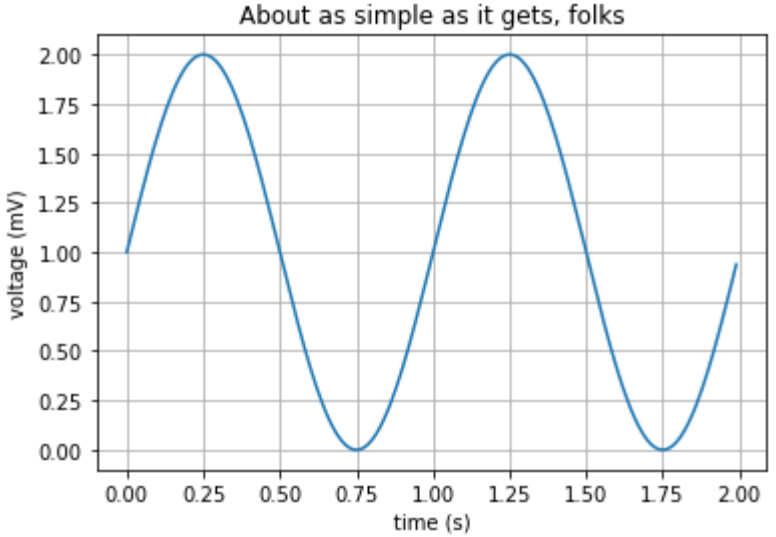
Matplotlib – библиотека для построения графиков и визуализации данных.

При работе с Jupyter необходимо перед использованием указать директиву ***%matplotlib inline***

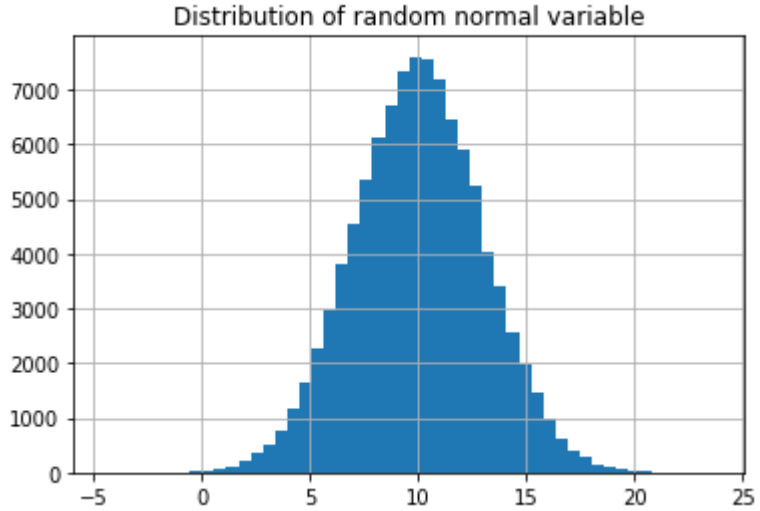
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()
fig.savefig("test.png")
plt.show()
```




```
x = np.random.normal(loc=10.0, scale=3.0, size=(100000,))
fig, ax = plt.subplots()
ax.hist(x, bins=50)
ax.set( title='Distribution of random normal variable')
ax.grid()
plt.show()
```



Библиотека Scikit-learn (sklearn)

Scikit-learn (sklearn) – библиотека алгоритмов машинного обучения, включающих классификацию, регрессию и кластеризацию. Особенности синтаксиса:

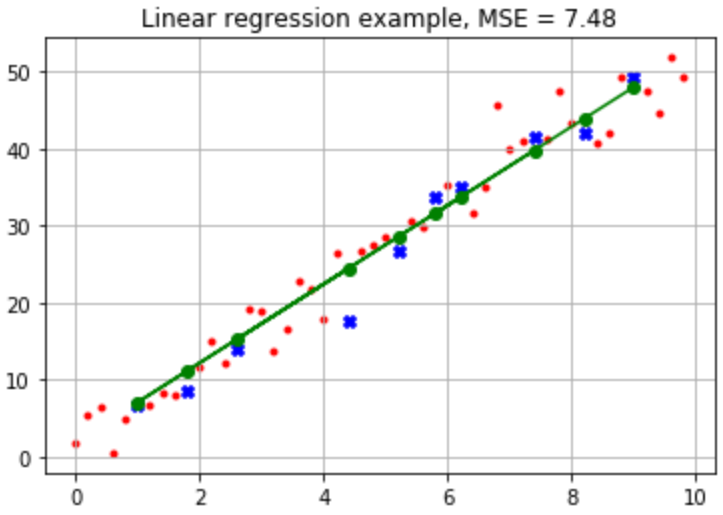
model = sklearn.КлассАлгоритма(гиперпараметры)
model.fit(train_data, train_labels)
pred = model.predict(test_data)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

X = np.arange(0.0, 10.0, 0.2)
Y = X*5 + np.random.normal(loc=2.0, scale=3.0, size=X.shape)
X = X.reshape(-1, 1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

regr = linear_model.LinearRegression()
regr.fit(X_train, Y_train)
Y_pred = regr.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)

fig, ax = plt.subplots()
ax.scatter(X_train, Y_train, marker=".", color="r", label="Train set")
ax.scatter(X_test, Y_test, marker="X", color="b", label="Test set")
ax.plot(X_test, Y_pred, marker="o", color="g", label="Predicted test")
ax.set(title=f'Linear regression example, MSE = {mse:.02f}')
ax.grid()
plt.show()
```



Спасибо за внимание!